Committee: ISO/IEC JTC 1/SC 22, Programming languages
Ballot Number: N 3754
Ballot Title: FCD 10967-3, Language independent arithmetic-Part 3: Complex integer and floating point arithmetic and complex elementary numerical functions
Source: JTC 1/SC 22 Secretary

National Body	Approval of the draft	Approval of the draft with comments	Disapproval with comments	Abstain	Comments
Australia					
Belgium					
Canada					
China	x				
Czech Republic	x				
Denmark				x	
Egypt					
Finland	x				
France	x				
Germany				x	
Ireland	x				
Italy	x				
Japan			X (acceptance of these reasons and appropriate changes to the text will change the vote to approval)		See below
DPR of Korea					
Republic of Korea	x				
Netherlands	x				
Romania					
Russian Federation	x				

Slovenia					
Switzerland				**X**	**No reply from expert**
Ukraine					
United Kingdom			**X (acceptance of these reasons and appropriate changes to the text will change the vote to approval)**		**See below**
United States				**x**	

National Body Comments

Japan

Japan disapproves the FCD document. There still remain many errors in the text, either technical or editorial. We must admit that most of these errors are minor, and we are ready to change our vote to "approve" if these errors are corrected.

==

Foreword, 2nd paragraph

The name of the document "ISO/IEC Directives, part 3" seems incorrect.
"JTC1" should be inserted after "ISO/IEC" to be consistent with the reference [1].

Introduction, "The content", 1st paragraph, 2nd line

There are two contiguous "specifications" in "specifications to specifications for operations". One of them should be deleted.

1.2(c)(d)(e)

The second sentence of these items are "This part neither requires nor excludes such data or operations.". The word "data" should be changed to "datatypes" in these sentences.

1.2(f)

This clause only speaks about "operations". The phrase "such data or operations" in the second sentence seems incorrect. It should be "such operations".

2, Note 1

This Note says that the Clause 8 requires a binding provided by an implementation. It is not easy to see this, since the word "binding" never appears in Clause 8. We suggest to give some explanations in this Note on the relationship between "binding" and "documentation".

3, Note 1

Here the word "annex" is capitalized, while many other appearances of "annex" are not capitalized. Please unify them to the appropriate form according to the convention for ISO/IEC standards.

4.1.2, set operators

The set operator "\times" appears twice; as the third operator in the list of set operators, and as an independent operator just following the list.

4.1.4, "overflow"

It seems that a noun is necessary just after "than".

4.1.4, "infinitary"

We cannot understand the meaning of the second half of the sentence.

What is referred to by "otherwise"? The phrase "from finite arguments"
should be changed to "at the finite argument point" for consistency.

4.1.4, 2nd last paragraph

The last sentence is very hard to understand for us. Something seems necessary after "specifications for". What is the relationship between "when it is appropriate ..." and "an appropriate continuation ..."?
The term "continuation value" is defined in 4.2, thus this is a forward reference. A section reference will help.

4.1.5, paragraph just after Note 2

One of two occurrences of "or part 2:" should be deleted.

4.1.5, "Floating point operations from part 1"

The "and" between "gtr_F" and "geq_F" should be deleted.

4.1.5, paragraph just after Note 5

An "are" should be inserted between "IEC 60559 and" and "used in this part".

4.2, "error"(2)

The word "context" should be changed to "contexts".

4.2, "normalised"

A section reference is written as "5.2 of part 1". This is inconsistent with a similar reference in "precision" where the same section is written as "clause 5.2 of part 1". We do not know whether "clause" should be or should not be written in such places. Please unify them according to the convention for ISO/IEC standards.

4.2, "signature"

We do not understand the naming convention of operations. For example, the name of the "cos" function taking an imaginary argument is "cos_i(F)".
Although this operation returns a real result of type F, this fact is not reflected in the operation name. On the other hand, the name of the "exp" function taking an imaginary argument is "exp_i(F)->c(F)" which contains the result type in its name. What is the principle under these namings?

5.1.2, "re_I" (for example)

The definition of "re_I" has a condition "if x \in I". We do not understand why such a condition is necessary. The intent might be to exclude infinities, but is it really necessary to exclude infinities from this definition?

5.1.2, "signum_I"

The values returned for infinity arguments are defined here. This is a violation to the following sentence in 5.1: "Integer datatypes conforming to part 1 often do not contain any NaN or infinity values, ... this clause has no specifications for such values as arguments or results.".

5.1.2, "quot_c(I)"

In this definition, a complex value is written as "x + (i.y)". These parentheses seem redundant. For example, see the definition of "mul_c(I)", where the same complex value is written as "x + i.y".

5.1.2, "mod_c(I),i(I)"

The definition given here is wrong. In the modulus, the real part of the argument "x" never goes to the imaginary part of the result. It should go to the real part of the result.

5.1.2, "residue_c(I),i(I)"

This definition is also wrong, just like "mod_c(I),i(I)".

5.1.2, "pad_I,i(I)"

This function should be similar to "mod_I,i(I)" and "residue_I,i(I)"
but there is a big difference from these two functions; the real part "x" is negated before passed to "pad_I" function. Is this intentional?

5,1,2, "pad_c(I),i(I)"

This definition is also wrong, just like "mod_c(I),i(I)" and "residue_c(I),i(I)".

5.1.2, "pad_c(I)"

The definition seems to have several errors. The second term should be subtracted from "x + i.y" not added to it. The function "round"
is used in the definition, but shouldn't this be changed to "ceiling"?

5.2.5, "plusitimes_c(F)"

This function takes two arguments of type F (real values). Why is the suffix of the operation name "c(F)"?

5.2.5, "add_i(F)" (for example)

The range of this function is "i(F) \union { (underflow), overflow }".
There must be some meaning conveyed by parentheses surrounding "underflow", but it is not explained anywhere.

5.2.5, "sub_F,i(F)" (for example)

The range of this function is defined to be "c(F)", but we think that a negative zero may result for a zero argument. In the definition of "conj_i(F)", the range of a similar result is written as "i(F \union {-0})".

Shouldn't the range of "sub_F,i(F)" also include "-0"?

5.2.5, "rounding_i(F)"

There is a TeX formatting error.

5.2.6, "signum*_c(F)"

In the requirements for this function, a special condition for "x + i.x"
is given requiring 1/sqrt(2) for both real and imaginary parts of the result. Is this condition consistent with similar conditions for other functions? For example, conditions for "polar*_c(F)" function are given in 5.2.7, for 30 degrees and 60 degrees cases, but not for 45 degrees case.

5.2.7

Some expressions are too wide for printing on A4 papers.

5.3.1.2 (for example)

The following sentence appears repeatedly in the definitions of functions:
"The requirements implied by these relationships and the requirements from part 2 shall hold even if there are no ... operations in any associated library for real-valued operations or there is no associated library for real-valued operations.". It is painful to read the same sentence several times. Isn't it possible to give this meta statement only once before giving definitions of specific functions?

5.3.1.3, "power_F->c(F)"

In the definition of "power_F->c(F)" function, a condition "|z/2| <= big_angle_u_F" is given several times. We think that the reference to "big_angle_u" in this part is not acceptable, since only radian-based trigonometrics are considered for complex cases. We suggest to change the condition to "|\pi.z| <= big_angle_r_F".

5.3.1.3, "power_i(F)" (for example)

In the definition, the real part is computed by "re_F(i.y)", but this function reference is not correct. Since the argument has the type "i(F)", the function name should be "re_i(F)" not "re_F". There are many occurrences of this error in the definitions of several functions.

5.3.1.3, "power*_c(F)"

A condition "y >= 0" is given for the cases "(z-0.5)/2 \in Z" and "(z-1.5)/2 \in Z" but not for the case "(z-1)/2 \in Z". Is this intentional?

5.3.1.3, "power_c(F)"

The condition given in the seventh alternative is wrong, or at the best redundant. It says "or x = -0" in its second term, but this is impossible since the fourth term is "x /= 0". The case "x = -0" is covered by the second alternative.

5.3.1.4, "sqrt_c(F)"

Certain cases are covered by more than one alternative. In other words, there are several overlappings in the conditions given for alternatives.
The case "x = +infinity and y = -0" is covered by the third and the sixth alternatives. The cases "x = -infinity and y = -0", "x = -0 and y = +infinity", and "x = -0 and y = -infinity" are also covered by two alternatives. The condition "x >= 0" in the fifth and the eighth alternatives should be "y >= 0".

5.3.1.5, "ln_c(F)"

The case "x \in {-0, 0} and y = -0" is doubly covered by the second and the third alternatives.

5.3.1.6, "logbase_i(F)"

One of two left parentheses before "re_F" should be deleted.

5.3.1.6

In the "associated library for real-valued operations" requirements, the term "this relationship" is singular. This should be changed to plural "these relationships", since there are at least two equations given.

5.3.2.5, Note

A space should be inserted between "when" and "x". The "a" should be deleted in "a zero". The alternative referred to in the Note says "x /= -0" contrary to the condition mentioned in the Note.

5.3.3

We consider that a sentence of the form "Note that ..." should not be used in the normative part of a standard.

7, 1st paragraph

The term "the present standard" looks strange. The term "this part" is often used in this document and refers to the same thing.

7, Note 1

In the first sentence, there is no mention to "binding standards".
We think that this sentence should also refer to "language or binding standards".

7, 3rd paragraph

We think that "to invoke an operation or access a parameter" should be changed to "to invoke an operation or {to} access a parameter" (two occurrences).

7, last paragraph

What is the meaning of "textual scope"?

B.1.1

The term "C99" is an informal one and seems inappropriate in this document.

B.1.2, 2nd paragraph

The first sentence is not grammatically correct. Change "no part" to "no parts" or "cover" to "covers".

B.2, 1st paragraph

The word "organisation" should be spelled as "organization".

B.4.1.2, 1st paragraph

In the first sentence, "is" should be changed to "are".

B.4.1.5

Is the position of the section header B.4.1.5 correct? The first sentence seems to belong to B.4.1.4.

B.4.1.5, 3rd paragraph

A "to" should be inserted after "while invalid with a specified continuation values".

B.4.2

The font of "Ada exceptions" should be Roman, not typeface, since it refers to a general notion in the Ada language, not a specific occurrence of a keyword.

B.5.3.1 (for example)

The phrase "because no programming language require them" is not grammatically correct. Change "language" to "languages", or "require"
to "requires". There are many occurrences of the same error.

B.5.3.2.4

The cycle of "tan" function is not 2.\pi, but is \pi.

B.5.3.2.5

The cycle is also \pi. The right-hand side of the cycle equation should be "cot" not "coth".

B.5.3.2.7

In the fourth equation, \pi/2 is multiplied by "i", but we do not understand this. We think that the "i." should be deleted.

B.5.3.2.9 (for example)

In the last equation, both plus and minus signs are allowed. Is this acceptable? We think that we only handle single-valued functions, and there would be no such freedom of sign. Are we wrong? By the way, what is the status of "Arccos" function (capitalized, Italic) in this document? Are they functions in the mathematics world?

B.5.3.3.2 (for example)

Since hyperbolic functions are only defined as references to trigonometric functions, we think that these many equations are not necessary for them.
Only the basic equivalences with trigonometric functions would suffice.

B.5.3.3.3

In the sixth equation, "= cos(-y)" also holds but is not mentioned. This seems inconsistent with similar equalities in "sinh", "tanh", ...

B.5.3.3.4

The cyle of "tanh" function is not i.2.\pi but is i.\pi.

B.5.3.3.5

The cycle is also i.\pi.

B.5.3.3.11

The condition given for the third equation is "if ...". This is obviously incomplete.

C, 1st paragraph of p.90

The word "examplified" is a typo. It should be "exemplified". The same error can also be found in the 2nd paragraph.

C.1, "plusitimes_c(I)"

The right parenthesis at the end of the expression should be deleted.

C.1, paragraph just after "min_seq_i(I)"

Defining the type of "xs" as "array of IINT" is not sufficient in Ada.
The index type of the array should also be defined.

C.1, 1st paragraph of p.94

The sentence beginning with "The parameter functions are constant ..."
is grammatically incorrect. Something should be inserted after the comma, or the comma should be replaced by a semicolon.

C.1, "eq_F,i(F)" (for example)

We do not think that these functions are defined by the Ada reference manual. Equality operators (equal and not-equal) are defined for two operands of the same type.

C.1, "power_c(F),I"

This function has a note "Not LIA-3". What is the intent to list such functions in the binding examples? There are many more occurrences of the same note in sections C.1 - C.5.

C.1, "ln_F->c(F)"

This function has two bindings. Readers may be unable to note this fact, since many functions with similar names are listed. Isn't it possible to show two expressions in an easily recognizable form? If the expressions are short, two expressions are shown in a line with an "or" delimiting them.
Some display form similar to this one would be desirable for long expressions.

C.1, "ln_F->c(F)"

The right parenthesis at the end of the second expression should be deleted.

C.1, "arcsinh_i(F)" (for example)

The function name is written as "ArcSinH". The same function in the Ada reference manual is "Arcsinh". The method of capitalization is different. We think that those names defined in the Ada reference manual should be used.

C.1, "arccosh_i(F)->c(F)"

The corresponding Ada function name is defined to be "ArcCosH", but why isn't this "ArcCosHc"? "arcsinh_i(F)->c(F)" is a quite similar function, and its corresponding Ada function is "ArcSinHc". The same comment also applies to "arcsech_i(F)->c(F)".

C.1, "convert_i(F)->i(F')"

The constant "i" is defined in a generic package. If there is more than one imaginary type, separate "i"s exist for each of these types. Therefore, "i" cannot be used without a qualifying package name.

C.1, "convert_c(F)->c(F')"

There is a TeX formatting error (position of star).

C.1, "convert_c(F")->c(F)"

The description of Ada wide characters as UCS-2/UTF-16 is not appropriate.
The Ada reference manual does not impose such strong constraints on the external representation of wide characters.

C.2, 6th paragraph of p.100

An "and" would be necessary between "CFLT" and "IFLT".

C.2, 7th paragraph of p.100

A sentence is erroneously split into two paragraphs.

C.2, "eq_c(I)" (for example)

How can these operators be defined? We think that introducing such operators in C is difficult. Since C does not have operator definitions, users cannot define operators by themselves. A compiler can introduce such operators, but extending the language in such a way is hard to justify.

C.2, "itimes_I->i(I)"

What is the type of "II"? "II" can be used for any imaginary types and thus its type is universal. Introducing such universal object in C would be difficult.

C.2, "eq_c(F)"

We think that the operator "==" is not language defined for two complex operands. Equality operators in C are defined for arithmetic or pointer types. Complex types are not considered arithmetic in C.

C.2, "exp_i(F)"

What is the meaning of a "dagger in parentheses"?

C.2, "ln_F->c(F)"

The right parenthesis at the end of the expression should be deleted.

C.2, "logbase_F->c(F)"

Is it intentional to use the function name "logbasec", while other similar functions are "clogbase"?

C.2, "arccos_i(F)->c(F)"

The function name is defined to be "acos". However, the corresponding function name for Ada is "ArcCosc". Why isn't the C function name "acosc"?

C.2, 2nd last paragraph of p.108

Some parts of the type descriptions are redundant. In the list of syntax, none of "y", "z" and "FXD" appears.

C.2, "imaginary_unit_c(I)"

Why are the parentheses doubled?

C.3, 4th paragraph

The hyphenation of "namespace" between "s" and "p" is not considered acceptable.

C.3, 7th paragraph

The sentence "CFLT is used below to designate one of the floating point datatypes." is not correct. "floating point datatypes" should be changed to "complex floating point datatypes".

C.3, "max_err_mul_c(F)"

The font of "<CFLT>" is not appropriate.

C.3, "signum_F"

What is the meaning of the question mark?

C.3, just after "polaru_F"

The phrase "where x and y are expressions of an imaginary or complex floating point type" is not quite precise. For example, the type of "u" is not defined. We think that the names "FLT", "IFLT" and "CFLT"
should also be defined for C++, and they should be used in such type descriptions.

C.3, "ln_i(F)->c(F)"

There is a TeX formatting error.

C.3, "logbase_c(F)"

This function should also have the comment "(note parameter order)".

C.3, "arccos_c(F)"

There are three "arccos" functions defined in 5.3.2.9. But, here only two of them are given bindings.

C.3, 2nd paragraph of p.117

Shouldn't "C print/scan family" be changed to "C printf/scanf family"?

C.4, next to "signum_i(I)"

Here, a binding for a complicated expression "mul_I(abs_I(y), signum_I(x))"
is given. Why is a binding necessary for such an expression?

C.4, 1st paragraph of p.124

A sentence is erroneously split into two paragraphs.

C.4, "power_c(F),I"

What is the meaning of the question mark?

C.4, "ln_F->c(F)"

The right parenthesis at the end of the expression should be deleted.

C.4, just after "arccsch_c(F)"

The phrase "and i of integer datatype" is not appropriate, since "i"
is not used anywhere.

C.5, "exp_i(F)"

The type of the operand in the reference notation is strange. Why is "i" multiplied in the operand position?

C.5, "ln_F->c(F)"

There is a TeX formatting error.

C.5, "ln_i(F)->c(F)"

There is a TeX formatting error. A right parenthesis should be inserted at the end of the expression.

C.5, "logbase_i(F)"

The comment "(note parameter order)" should also given to these functions.

C.5, "arcsin_F->c(F)"

The function name is defined to be "asin", but the corresponding name in Ada or C has "c" at the end of the name. What is the reason for this difference?

C.5, "arctan_i(F)"

The function name is "atanx". Why isn't it "atanc" like in other languages?

C.5, last paragraph of p.135

"but n1 and n1 are ..." should be changed to "but n1 and n2 are ...".

D

Is there any policy in selecting references? If the intent is to list every major language in the reference, C# which is now an international standard should also be listed.

D

The note in [15] should also be given to [7] and [8].

D

The reference [9] will soon be withdrawn from the list of international standards.

D

The name of the journal in the reference [52] seems incorrect. It would be "SIAM Journal on Numerical Analysis".

United Kingdom

UK COMMENTS ON ISO/IEC FCD 10967-3

UK1) General
The standard is too big with too many functions. The ballot for the previous draft closed 22 months ago. With WG11's limited available effort, a smaller, and thus more rapid, standard is advisable.

UK2) General
The set of proj functions for Ada, which are not replicated for other languages, look particularly odd.

UK3) General
Language implementors and standardisers may not find the suggestion that FCD 10967-3 adds lots of new functions and features appealing, particularly when the facilities they provide are already present in the language in other, sometimes only very slightly different, forms.

UK4) General/Technical
The accuracy of evaluating a computer function f(x) where x is some floating point value should be judged not by its closeness to the value of the corresponding mathematical function and value F(x), but by its closeness to the value of F(x') where
    x' is some value such that abs(x-x') < 0.5 * ulp
and
    ulp is the value of a unit in the last significant place of x

This reflects the fact that a floating point (complex) value really represents one of a range of values. J H Wilkinson used this idea in his scheme of 'backwards error analysis'.

To take an extreme case, a computer with floating point with 10 significant digits should not need to find the remainder of exp(100) after dividing by 2pi when asked to calculate sin(exp(100)) but instead give any value between -1 and 1, or better still signal an error (value has no meaning).

UK5) General/Technical
It is unnecessary to define the value of a function when one of its arguments is not a number (NaN, or infinity, or overflow, etc). These values can sensibly be left undefined.

UK6) General/Technical
GB cannot imagine a use for a complex integer data type to which so much space is devoted, nor indeed an imaginary (as distinct from a complex) floating point type. Since 1991 Fortran has provided facilities which would allow the user to create their own Gaussian integer type, and yet there has not been a great proliferation of such packages (or applications using such packages) in the public domain - nor has their been any clamour for intrinsic (compiler) support for such a type.

UK7) General/Editorial
The 'not LIA-3' notation which appears opposite some operations in each of the language bindings is puzzling. If the operation is not in the standard what is the purpose of the description in the binding?

UK8) General/Editorial
The notation is confusing to the point of opacity. It would have been easier to understand (and use just as much paper) to spell out the argument and result data types rather than forcing those unfamiliar with all the multitudinous functions to hunt through document to find them.

UK9) General/Editorial
In particular, saying that "*" was for syntax that was "already provided" (in contrast to "†") is very strange since, in many cases, the standard in question provides that syntax. What it does not provide is the datatype for the already-provided syntax to act upon!

Perhaps the wording for "*" is for where the language provides not only the syntax, but also the operations?

UK10) Technical
Fortran provides two COMPLEX datatypes, not one:

```
  COMPLEX(KIND(0.0))  aka COMPLEX
  COMPLEX(KIND(0d0))
```

UK11) Technical
The choice of IEEE_RINT for the "floor", "rounding" and "ceiling" operations is, I believe, incorrect, simply because it is (and should be) only applicable to IEEE machines not in general.

In particular, for "rounding", ANINT would seem to be a more obvious choice.

UK12) Technical
Does

```
  CMPLX(x)**y
```

not implement what FCD 10967-3 suggests should be POWER(x,y)?

UK13) Technical
Does

```
  SQRTC(x)
```

not implement what FCD 10967-3 suggests should be SQRT(CMPLX(x)).

UK14) Technical
Does

```
  LOGC(x)
```

not implement what FCD 10967-3 suggests should be LOG(CMPLX(x)).

UK15) Technical
There are two forms provided for "ln[F->c(F)](x)", the first of which is LOGC(x) the second of which is a complicated formula using the variable I.
In Fortran, the mathematical "i" is written as "(0,1)". For the maths "i", use italic lower-case to avoid confusion.

In any case [see technical comment C.6a] it's already provided as LOG(CMPLX(x)).

UK16) Technical
FCD 10967-3 states that "Arithmetic value conversions for complex in Fortran is done via explicitly converting the real and imaginary part (and recomposing the results)."
This does not appear to be true. The CMPLX intrinsic may be used to convert one complex type to another (or indeed any type to complex).

UK17) Editorial
'(not LIA-3)' on page 124 should be 'Not LIA-3' for consistency with other appearances, if they are to remain.

UK18) Editorial
There is a '?' by the <complex> ** <integer> operation, on page 124. If this is meant to denote uncertainty, then this operation is indeed defined in Fortran and has been since the 1966 standard.

UK19) Editorial
Why is there a query at the C++ signb operation on page 114 ?

UK20) Editorial
As the deadline for this vote is October, it seems very likely that by then the revised Fortran standard will, if not published, be on the verge of publication. Therefore '1539-1:1997' could be changed to '1539-1:2004' on pages 118 and 138. This would have no effect on the binding so far as I can see.

UK21) Editorial
There is a "(" missing in the fifth MODULO line on page 120.

UK22) Editorial
Spelling "cosec" as "CSC" is ugly.
"ACSCHC" is even worse, and is equivalent to ACSCH(CMPLX(x)) anyway.

END OF UK COMMENTS